UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Boštjan Kodre

Prenos namizne igralniške aplikacije na mobilno platformo

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: dr. Igor Rožanc

Ljubljana, 2009

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Boštjan Kodre,

z vpisno številko 63040230,

sem avtor diplomskega dela z naslovom:

Prenos namizne igralniške aplikacije na mobilno platformo

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  dr. Igor Rožanc
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 3.7.2009 Podpis avtorja/-ice:

Zahvala

Zahvaljujem se mentorju dr. Igorju Rožancu za strokovne nasvete in usmeritve pri izdelavi diplomskege naloge.

Zahvaljujem se tudi sodelavcem za vzpodbudo in vso pomoč pri nastajanju diplomskega dela.

Posebna zahvala gre vsem, ki so mi stali ob strani vsa leta študija in me ob tem podpirali ter vzpodbujali. Družini, vsem prijateljem ter punci Teji, od katerih sem prejel pozitivno energijo in zaupanje.

Kazalo

Slike

Seznam uporabljenih kratic in simbolov

bluetooth	brezžična tehnologija za povezovanje naprav na kratkih razdaljah
CIL	ang. Common Intermediate Language skupen vmesni jezik
CLI	ang. Common Language Infrastructure jezikovno neodvisna platforma
CLR	ang. Common Language Runtime
CRM	ang. customer relationship management upravljanje odnosov s strankami
GDI	ang. Graphics Device Interface grafični vmesnik naprave
GDI+	ang. Graphics Device Interface Plus naslednik gdi-ja
IIS	ang. Internet Information Services internetni informacijski strežnik
RFID	ang. Radio Frequency IDentification identifikacija z radijskimi valovi
SOAP	ang. Simple Object Access Protocol standard za spletne storitve, ki temelji na XML
wi-fi	brezžično lokalno omrežje
WSDL	ang. Web Services Description Language opisni jezik spletnih storitev
XML	ang. Extensible Markup Language razširljiv označevalni jezik

Povzetek

Poznavanje načina igre igralcev na srečo je v igralništvu zelo pomembno. V ta namen je bila razvita namizna aplikacija za sledenje igre gostov na igralnih mizah, ki pa zaradi pomankanja mobilnosti ni zadoščala za učinkovito sledenje. Zato je bilo potrebno prenesti to namizno aplikacijo na mobilno platformo. V diplomskem delu smo se osredotočili predvsem na težave, ki so se pojavljale pri prenosu, jih opisali in podali njihovo rešitev. Povezane so pretežno z izdelavo grafičnega vmesnika ter z vzpostavitvijo ustrezne komunikacije med strežnikom in mobilno aplikacijo. Ker je namizna aplikacija razvita v .NET ogrodju, mobilna aplikacija pa v .NET kompaktnem ogrodju so težave, predvsem odraz razlik med navedenimima ogrodjema. Rezultat dela je delujoča mobilna aplikacija, ki teče na dlančniku z operacijskim sistemom Windows Mobile in nameščenim .NET kompaktnim ogrodjem.

Ključne besede:

dlančnik, mobilne aplikacije, .NET kompaktno ogrodje, .NET ogrodje

Abstract

Knowing the importance of the players is very important in gambling industry. For this purpose was developed desktop application for tracking the game of guests on the gaming tables. This application was not sufficient for effective tracking due to lack of mobility, therefore was necessary to transfer it to the mobile platform. In the thesis I focused mainly on problems that have occurred during the transfer, I described them and their's solutions. They are mostly connected with making graphic interface, and communication between server and mobile application. Since the desktop application is developed with .NET Framework and mobile application with .NET Compact Framework are the problems I had during development reflection of differences between those Frameworks. The result of my work is mobile application, which is running on palmtop with Windows Mobile operating system and .NET Compact Framework installed.

Key words:

palmtop, mobile aplications, .NET Compact Framework, .NET Framework

Poglavje 1

Uvod

Želja po mobilnosti je pri ljudeh prisotna že od nekdaj. Tako kot v drugih panogah se je tudi v računalništvu povečevala skladno s tehnološkim razvojem. Prvi poskusi po mobilnosti so pripeljali do iznajdbe prenosnega računalnika. Prenosni računalniki kot taki niso zadostili vsem potrebam po mobilnosti, saj so preveliki za uporabo med premikanjem, kot odgovor na ta problem so se pojavili dlančniki, katerih uporabnost se je povečevala z razširjenostjo brezžičnih omrežij. V zadnjem času postajajo tudi mobilni telefoni vse boljši in podpirajo vedno več funkcionalnosti, ki so bile prej značilne samo za dlančnike. Tako so določeni temelji za izdelavo mobilnih aplikacij, ki tečejo na dlančniku ali mobilnem telefonu ter preko brezžične komunikacije komunicirajo z drugimi programi (spletnimi storitvami), katerih naloga je shranjevanje podatkov, ki so posredovani z mobilne aplikacije, ter posredovanje podatkov mobilni aplikaciji.

V igralništvu je podobno kot v drugih vrstah turizma zelo pomembno privabljati nove goste ter obdržati obstoječe. Zato je potrebno goste stimulirati, da ponovno pridejo v igralnico. Ker vsi gostje ne prispevajo enako k prihodkom igralnice, je potrebno temu prilagoditi tudi stimulacijo gosta: zapravljivejše goste želimo močneje stimulirati. Zapravljivost gosta pa je brez posebnega sistema skoraj nemogoče določiti. Zaradi tega je bila v podjetju Hit d.d. razvita namizna aplikacija PRAUŽ (Player Rating z Avtomatskim Urejevalnikom Žetonov), ki omogoča merjenje porabe gosta pri tako imenovanih živih igrah[1], ki potekajo na igralnih mizah.

Pri uporabi obstoječe aplikacije PRAUŽ se je pojavila potreba po mobil-

[1]Ime živa igra izhaja iz tega, da človek skrbi za potek igre in se igra ne izvaja sama kot na avtomatih. Primeri živih iger so: ameriška ruleta, black jack in druge.

nosti, ki pomeni predvsem to, da je lahko uslužbenec igralnice fizično sledil gostu od mize do mize in beležil njegovo igro. Tej potrebi po mobilnosti je sledila izdelava mobilne aplikacije z istimi funkcionalnostmi kot jih ima obstoječa namizna aplikacija. Razvoj te aplikacije je jedro pričujoče diplomske naloge. Pri tem smo se osredotočili predvsem na težave pri razvoju, ki so specifične za razvoj mobilnih aplikacij.

Poglavje 2

Opis dlančnika in uporabljenih orodij ter tehnologij

2.1 Dlančnik

Dlančnik (slika 1) je računalnik v malem [11]. Njegovo majhnost izraža že

Slika 1: Dlančnik z operacijskim sistemom Windows Mobile 6

sama beseda dlančnik, ki kaže na to, da ga lahko držimo v dlani. Za prvi dlančnik se šteje Casio PF-3000, ki je bil predstavljen leta 1983, čeprav se je ime dlančnik uveljavilo šele leta 1992, ko je na trg prišel Apple Newton. Njihova popularnost je naraščala do leta 2003, ko jih je bilo prodanih kar 12 milijonov. Po tem letu pa se je pričel upad prodaje na račun pametnih telefonov. Dandanes se jih uporablja predvsem v skladiščih, gostinstvu, zava-

rovalništvu in zdravstvu, skratka povsod, kjer je potrebno na terenu zbrane podatke vpisovati v računalnik. Tipičen dlančnik ima zaslon na dotik, je brez tipkovnice (nadomešča jo navidezna tipkovnica), za povezljivost pa poskrbita wi-fi ter bluetooth.

2.2 Windows Mobile

Windows Mobile je kompakten operacijski sistem namenjen dlančnikom, pametnim telefonom ter drugim mobilnim napravam [16]. Izhaja iz družine operacijskih sistemov Windows CE. Zasnovan je tako, da je podoben namiznim različicam operacijskega sistema Windows. Serija operacijskih sistemov Windows Mobile se je začela leta 2000 z Pocket Pc 2000 [12], ki je bil izdan za 3 različne arhitekture centralnih procesnih enot: SH-3, MIPS in ARM. Vseboval je aplikacijo za prepoznavo pisave, Pocket Office, Pocket Internet Explorer in Windows Media Player. Njegova največja omejitev je bila ločljivost zaslona, saj je podpiral le 240 x 320 točk. Z verzijo Pocket Pc 2002 je bila dodana podpora sinhronizaciji map, navideznemu zasebnemu omrežju (ang. virtual private network - VPN) in terminalskim storitvam, izboljšana pa je bila tudi združljivost z pametnimi telefoni. Problem nizke ločljivosti zaslona so rešili šele z Windows Mobile 2003, ko so podprli ločljivosti 240 x 240 in 480 x 480, izboljšali pa so tudi povezljivost s podporo bluetooth-a ter wi-fi-ja. V letu 2005 je izšel Windows Mobile 5 z vgrajenim .NET kompaktnim ogrodjem, izboljšano združljivostjo z komunikacijsko infrastrukturo ter interakcijo z Microsoft Exchange Server. Šesta izvedba operacijskega sistema ni prinesla večje prenove razen boljše stabilnosti sistema.

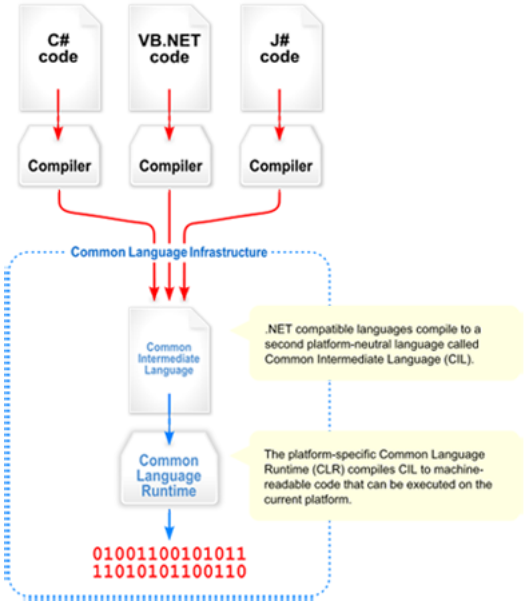
2.3 .NET kompaktno ogrodje

.NET kompaktno ogrodje [1] (ang. .NET Compact Framework) je okleščena verzija .NET ogrodja, narejena za mobilne naprave, kot so dlančniki, pametni telefoni, itd. Za zmanjšanje porabe prostora so nekatere razrede odstranili iz .NET ogrodja, nekatere pa so okrnili. Zaradi specifičnih zahtev mobilnih naprav pa so nekateri razredi tudi dodani, naprimer `InputPanel`. .NET kompaktno ogrodje si z .NET ogrodjem deli isto zasnovo. Oba imata veliko število knjižnic, ki rešujejo pogostejše probleme pri programiranju, imata pa tudi virtualni stroj - CLR, ki skrbi za izvajanje programov, upravljanje z napakami in spominom.

2.3.1 Arhitektura .NET ogrodja in .NET kompaktnega ogrodja

2.3.1.1 Jezikovno neodvisna platforma

Jedro .NET ogrodja je znotraj jezikovno neodvisne platforme (slika 2)[7] (ang. Common Language Infrastructure - CLI). Njen namen je, kot že samo ime pove, zagotavljanje jezikovno neodvisne platforme za razvijanje in izvajanje, vključno s funkcijami upravljanja izjem (ang. exception handling), zbiranja smeti (ang. garbage collection), varnosti in interoperabilnosti. Microsoftova implementacija CLI-ja se imenuje Common Language Runtime.



Slika 2: Grafični prikaz CLI

2.3.1.2 Programske komponente

Programska komponenta (ang. assembly) je delno prevedena izvorna koda knjižnice [6]. Obstajata dva tipa: procesne (EXE) in knjižnične programske komponente (DLL). Procesna programska komponenta predstavlja proces, ki uporablja razrede iz knjižnične programske komponente. Programska komponenta je lahko sestavljen iz ene ali več datotek (modulov). Ker je mogoče uporabiti več različnih jezikov pri ustvarjanju modula, je tehnično mogoče uporabiti več programskih jezikov za kreiranje programske komponente recimo

Visual Basic in C#. Iz .NET jezikov se ob prevajanju ustvari programska komponenta, ki vsebuje kodo v skupnem vmesnem jeziku (CIL). CIL je najnižji nivo s človeku berljivim jezikom znotraj jezikovno neodvisne platforme. Virtualni stroj skrbi za izvajanje programske komponente, tako da sproti interpretira CIL.

2.3.1.3 Metapodatki

Metapodatki (ang. metadata) so binarni podatki, ki opisujejo program, ki je shranjen bodisi v CLR prenosni izvršljivi (ang. Portable Executable - PE) datoteki bodisi v pomnilniku [10]. Metapodatki opisujejo vse razrede ter posamezne dele razredov. Metapodatki metod vsebujejo opis metod, tip podatka, ki ga metoda vrača ter tipe vseh parametrov metode. Metapodatki se ustvarijo ob prevajanju programske kode.

2.3.1.4 Varnost

.NET ogrodje vsebuje več mehanizmov za varovanje virov in kode pred nepooblaščeno kodo in uporabniki [3]:

- ASP.NET spletna aplikacijska varnost (ang. web application security) omejuje dostop do strani s pomočjo primerjave overjene poverilnice z dovoljenji na Microsoft Windows NT datotečnem sistemu ali z XML datoteko, ki vsebuje seznam pooblaščenih uporabnikov, pooblaščenih vlog ter pooblaščenih HTTP naslovov.

- Varnost dostopa kode (ang. code access security) uporablja dovoljenja za omejevanje dostopa do virov in operacij. Računalniškim sistemom pomaga pri zaščiti pred zlonamerno kodo, ter pomaga zagotavljati varno izvajanje kode.

- Na vlogah temelječa varnost (ang. role-based security) zagotavlja informacije, ki so potrebne pri odločitvi o tem, katere akcije dovoliti uporabniku. Te odločitve lahko temeljijo na identiteti uporabnika, vlogi članstva, ali obeh.

2.3.1.5 Knjižnica razredov

Knjižnica razredov (ang. class library) vsebuje razrede, vmesnike in vrednostne tipe za pospešitev in optimizacijo razvojnega procesa ter za dostop do sistemskih funkcionalnosti [9]. Zaradi olajšanja interoperiabilnosti med jeziki

so tipi v .NET ogrodju skladni s specifikacijo skupnega jezika (ang. Common Language Specification - CLS), zato lahko uporabljamo katerikoli jezik, ki ima prevajalnik skladen z CLS. Tipi .NET ogrodja predstavljajo osnovo, na kateri so zgrajene .NET apilkacije, komponente in gradniki. .NET ogrodje vključuje tipe, ki opravljajo naslednje naloge:

- predstavljajo osnovne podatkovne tipe in izjeme,
- ovijajo podatkovne strukture,
- opravljajo vhodno/izhodne (I/O) operacije,
- omogočajo dostop do informacij o naloženih tipih,
- izvajajo varnostna preverjanja ter
- zagotavljajo dostop do podatkov, bogat grafični uporabniški vmesnik (GUI).

Za poimenovanje razredov v knjižnjici razredov se uporablja sintaksa z piko, ki predstavlja hierarhijo. Ta tehnika združuje sorodne tipe v imenski prostor (ang. namespace), zato da jih lažje iščemo in se nanje sklicujemo. Imenski prostor je prvi del polnega imena (do prve pike), zadnji del pa je ime tipa.

2.3.1.6 Upravljanje s pomnilnikom

CLR .NET ogrodja sam skrbi za upravljanje s pomnilnikom [2] (ang. Memory management). V ta namen CLR neprestano dodeljuje pomnilnik instancam .NET tipov (objektov) iz zaloge pomnilnika, ki je upravljan s strani CLR-ja. Dokler obstaja sklic na objekt, ki je lahko neposredni ali preko grafa objektov, se šteje, da je objekt v uporabi v CLR. Ko ne obstaja noben sklic na objekt, objekta ni mogoče več doseči zato postane odvečen. Tak objekt še vedno drži pomnilnik, ki mu je bil dodeljen, dokler smetar (ang. garbage collector) ne zaseže pomnilnika, ki pripada temu objektu. Smetar se požene šele tedaj, ko je porabljena določena količina pomnilnika ali ko je veliko povpraševanje po pomnilniku v sistemu. Vsaka .NET aplikacija ima niz korenov, ki so kazalci na objekte v zalogi pomnilnika, ki je upravljan s strani CLR-ja. Ko se požene smetar, ustavi aplikacijo in za vsak objekt v korenu rekurzivno pregleda vse objekte, ki so dosegljivi iz korena, ter jih označi za dosegljive, ostale pa označi kot smeti. Pomnilnik uporabljen s strani objektov, ki so označeni kot smeti, se šteje kot prazen prostor. Ker na ta način nastaja med prej strnjenimi objekti prazen prostor, smetar strne skupaj dosegljive objekte. Ko smetar

konča z delom ponovno požene izvajanje aplikacije. .NET smetar razvršča objekte v generacije. Novo kreirani objekti pripadajo generaciji 0, objekti ki preživijo prvo izvajanje smetarja pripadejo generaciji 1, objekti generacije 1, ki preživijo ponovno izvajanje smetarja, pa postanejo člani generacije 2. Pogostost izvajanja smetarja nad posameznimi objekti je bolj pogosta, če so nižje generacije, saj se šteje da imajo krajšo življensko dobo kot tisti, ki so višje generacije. Na ta način je močno izbojšano delovanje smetarja saj se število objektov, ki jih mora pregledovati, drastično zmanjša.

2.4 Razvojna platforma Mono

Mono je odprtokodna razvojna platforma, ki temelji na Microsoft .NET ogrodju in omogoča izdelavo Linux in medplatformnih aplikacij, omogoča pa tudi izvajanje .NET aplikacij v operacijskem sistemu Linux [15]. Mono je sestavljen iz sledečih komponent:

- C# prevajalnik; trenutno je popolno podprto prevajanje C# 1.0 in 2.0 (ECMA standard), vsebuje pa tudi veliko funkcij od C# 3.0
- Mono izvajalno okolje (ang. Mono runetime); Mono izvajalno okolje izvršuje ECMA CLI, ter nam zagotavlja sprotni (ang. just-in-time) prevajalnik, vnajpejšni (ang. Ahead-of-Time) prevajalnik, nalagalnik knjižnjic, zbiranje smeti in sistem upravljanja z nitmi.
- Osnovna knjižnica razredov (ang. Base Class Library); Zagotavlja celovit sklop razredov, ki so združljivi z Microsoftovimi razredi .NET ogrodja, ki zagotavljajo temelje za izdelavo aplikacij.
- Mono knjižnica razredov (ang. Mono Class Library); Mono zagotavlja tudi številne razrede, ki razširjajo Microsoftovo osnovno knjižnico razredov. Ti razredi so izredno uporabni pri izdelavi Linux aplikacij. Podpirajo funkcionalnosti kot so delo z: OpenGl, Zip datotekami, LDAP-om, ...

2.5 Grafični vmesnik

Grafični vmesnik (ang. Graphics Device Interface - GDI) je programski vmesnik v operacijskem sistemu Windows, ki je zadolžen za predstavitev grafičnih objektov ter za pošiljanje na izhodne naprave kot so monitorji in tiskalniki [8].

Zadolžen je tudi za naloge, kot so risanje linij in krivulj, izrisovanje pisav in upravljanje z barvnimi paletami. Največja prednost pred bolj neposrednimi metodami dostopa do naprave je odmišljanje ciljne naprave. S prihodom Windows Xp je uporaba GDI opuščena, saj je napisan nov programski vmesnik za delo z grafičnimi objekti GDI+. GDI+ zagotavlja strojno pospeševanje z direktno interakcijo z grafično napravo v imenu aplikacije. GDI+ podpira mehčanje robov pri 2D grafiki, koordinate so izražene z realnimi števili, senčenje gradienta, kompleksnejše upravljanje z potmi, podporo za moderne grafične formate kot so JPEG in PNG,... Za predstavitev barv so uporabljene ARGB vrednosti, kar omogoča izris prosojnih barv.

2.6 Programski vmesnik .NET Remoting

.NET Remoting je programski vmesnik za medprocesno komuniciranje [4][5]. Omogoča izgradnjo objektov, ki so dosegljivi iz drugih procesov na istem računalniku ali iz kateregakoli računalnika, ki je povezan v omrežje. .NET Remoting uporablja abstraktni pristop k medprocesni komunikaciji, da loči objekt od specifičnega klienta, strežnika in mehanizma za komunikacijo. Pri uporabi .NET Remotinga za izdelavo aplikacije potrebujemo:

- oddaljen objekt,
- gostiteljsko aplikacijo, ki posluša zahteve po objektu in
- odjemalčevo aplikacijo, ki povprašuje po objektu.

.NET Remoting naredi sklic na oddaljen objekt, ki je dostopen odjemalčevi aplikaciji. Odjemalčeva aplikacija uporablja objekt kot bi bil lokalen, vendar se vse izvajanje dogaja na strežniku. Oddaljeni objekt se identificira z aktivacijskim url-jem, ob povezavi na url pa se objekt inicializira. .NET Remoting ustvari poslušalca oddaljenega objekta, ko strežnik prepozna kanal, ki je uporabljen za povezavo do oddaljenega objekta. Na odjemalcu .NET Remoting infrastruktura naredi proxy, ki deluje kot nekakšna pseudo instanca oddaljenega objekta. Ta ne implementira funkcionalnosti oddaljenega objekta, temveč samo predstavlja vmesnik. Zato mora .NET Remoting infrastruktura poznati javni vmesnik oddaljenega objekta pred uporabo oddaljenega objekta. Vsi klici metod oddaljenega objekta so zaporedno objavljeni v tok bajtov in so prenešeni po komunikacijskem kanalu do naslovnega proxy objekta na strežniku. Na strežniku proxy prebere tok bajtov in sproži metodo v imenu odjemalca. Rezultat se zaporedno uredi ter pošlje do odjemalca, kjer proxy prebere rezultat ter ga posreduje klicoči aplikaciji.

2.7 Spletne storitve

Spletna storitev je programski sistem za podporo medobratovalnosti med napravami preko omrežja [13][14]. Spletne storitve so pogosto le spletni programski vmesniki (ang. Application Programming Interfaces - API), ki so dosegljivi preko omrežja in se izvajajo na oddaljenem sistemu, ki gostuje to spletno storitev. Lastnosti spletnih storitev so:

- samovsebovanost: Na odjemalcu je za delovanje potreben le programski jezik z podporo http-ju in xml-ju. Spletni strežnik in odjemalec sta lahko implementirana v različnih okoljih.

- samoopisanost: Odjemalec in strežnik morata poznati samo format in vsebino sporočila. Definicija formata sporočila potuje skupaj s sporočilom, tako da ni potrebe po zunanjih metapodatkih.

- modularnost: Enostavne spletne storitve se lahko združijo v bolj zapletene spletne storitve z uporabo tehnik delovnega toka (workflow) ali s klici nižjih spletnih storitev.

- platformna neodvisnost: Temeljijo na nizu odprtih na XML osnovanih standardih za doseganje medobratovalnosti med spletnimi storitvami in odjemalci na različnih vrstah računalniških platform ter programskih jezikov.

Za pošiljanje sporočil je uporabljen XML, ki sledi standardu SOAP. Za opis operacij, ki jih ponuja spletna storitev, se uporablja WSDL. Na podlagi WSDL opisa spletne storitve nekatera orodja omogočajo avtomatsko izdelavo odjemalčeve programske kode.

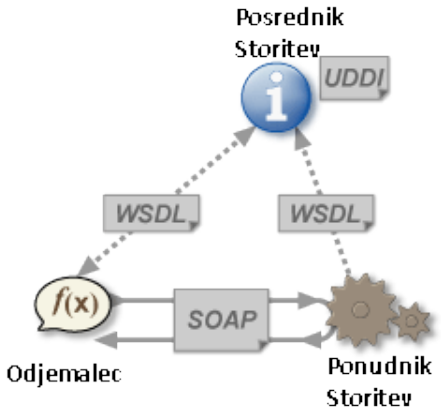
2.7.1 Vloge storitev in stiki med njimi

Mrežna komponenta lahko igra v arhitekturi spletnih storitev eno ali več temeljnih vlog (slika 3) kot:

- ponudnik storitev: Ustvari spletne storitve in lahko objavi razpoložljivost njihovih z WSDL opisanih spletnih storitev v storitveni register.

- posrednik storitev: Registrira in ureja objavljene servise, ter omogoča iskanje servisov. Storitveni register je posrednik storitev za spletne storitve, ki so opisane z WSDL.

- odjemalec: Preko posrednikov storitev pridejo do WSDL opisa storitve in nato kličejo storitev pri ponudniku storitev.



Slika 3: Delovanje spletnih storitev

Poglavje 3

Predstavitev obstoječe namizne aplikacije PRAUŽ

3.1 Opis problema

Kot v vsakem podjetju, ki ima namen izboljševati svoje poslovanje in odnos do gostov, se je tudi v podjetju Hit d.d. pojavila želja po tem, da bi izmerili učinkovitost promocijskih akcij ter ocenili, kateri igralci so za podjetje najpomembnejši. Pot do realizacije te želje je uvedba sledenja igralcem, na podlagi katerega bi dobili podatke o igri igralca, ter ga nato temu primerno tudi obravnavali. Zaradi tega je podjetje kupilo produkt[1], ki omogoča sledenje igralcem, vendar le pri igrah na avtomatih. Tako so postali igralci na igralnih mizah spregledani, čemur je sledila odločitev, da se del za pokrivanje sledenja na igralnih mizah razvije interno v podjetju. Aplikacija mora pokrivati naslednje zahteve:

- omogočati mora sledenje klubskim igralcem,
- podpirati mora sledenje na ruleti ter pri igrah s kartami (Poker, Black Jack, Texas Holdem Poker, ...) in
- omogočati mora prepis podatkov v sistem CRM[2].

[1]Njegova primarna funkcija je obračunavanje stanja denarja na avtomatu in zbiranje podatkov o igri igralcev. To je integriran informacijski sistem, ki stalno spremlja igralne avtomate, druge igralne naprave in igro strank.

[2]To je sistem za upravljanje s strankami, v katerem se zbirajo vsi podatki o gostih. Na podlagi teh podatkov se izvajajo razne analize in primerjave strank, ki nam služijo kot pripomoček za pripravo trženjskih in drugih akcij.

3.2 Zajem zahtev

3.2.1 Slovar izrazov

Igralec Oseba, ki je prišla v igralnico z namenom igranja iger na srečo.

neznani igralec - NN Oseba, ki še ni identificirana in igra katerokoli igro na srečo.

Sledilec Oseba, ki je zaposlena v igralnici in izvaja sledenje igre gostov.

Inšpektor Oseba, ki je zaposlena v igralnici z nalogo nadziranja poteka iger.

Igralniška recepcija - IR Igralniška recepcija je aplikacija za beleženje gostov ter njihovih obiskov igralnice. Zbirka podatkov igralniške recepcije se uporablja kot skupno jedro vseh aplikacij, ki so povezane z gosti. V diplomski nalogi uporabljamo iz igralniške recepcije podatke o:

- delavcih,
- lokacijah,
- valutah
- osebah,
- slikah oseb,
- ocenah oseb,
- ocenah oseb za posamezen dan,
- vsakokratnih slikah oseb,
- albumskih slikah oseb,
- vstopih igralcev,
- klubskih članstvih in
- nivojih kluba.

igralna miza - IM Miza na kateri se odvijajo igre kot so: ameriška ruleta, francoska ruleta, poker, black jack,...

avtomatski urejevalec žetonov - AUŽ Avtomatski urejevalec je naprava za razvrščanje igralnih žetonov glede na njihovo barvo. Med razvrščanjem šteje število razvrščenih žetonov po posameznih barvah.

povprečna stava Seštevek vseh stav deljen s številom stav.

denominacija žetona Vrednost barvnega žetona, izražena v evrih.

igralni dan - IDAN Čas med 07:00 zjutraj in 07:00 zjutraj naslednjega koledarskega dne.

meritev Meritev je zaključena igra igralca v igralnem dnevu, sestavljena je iz več zapisov meritve.

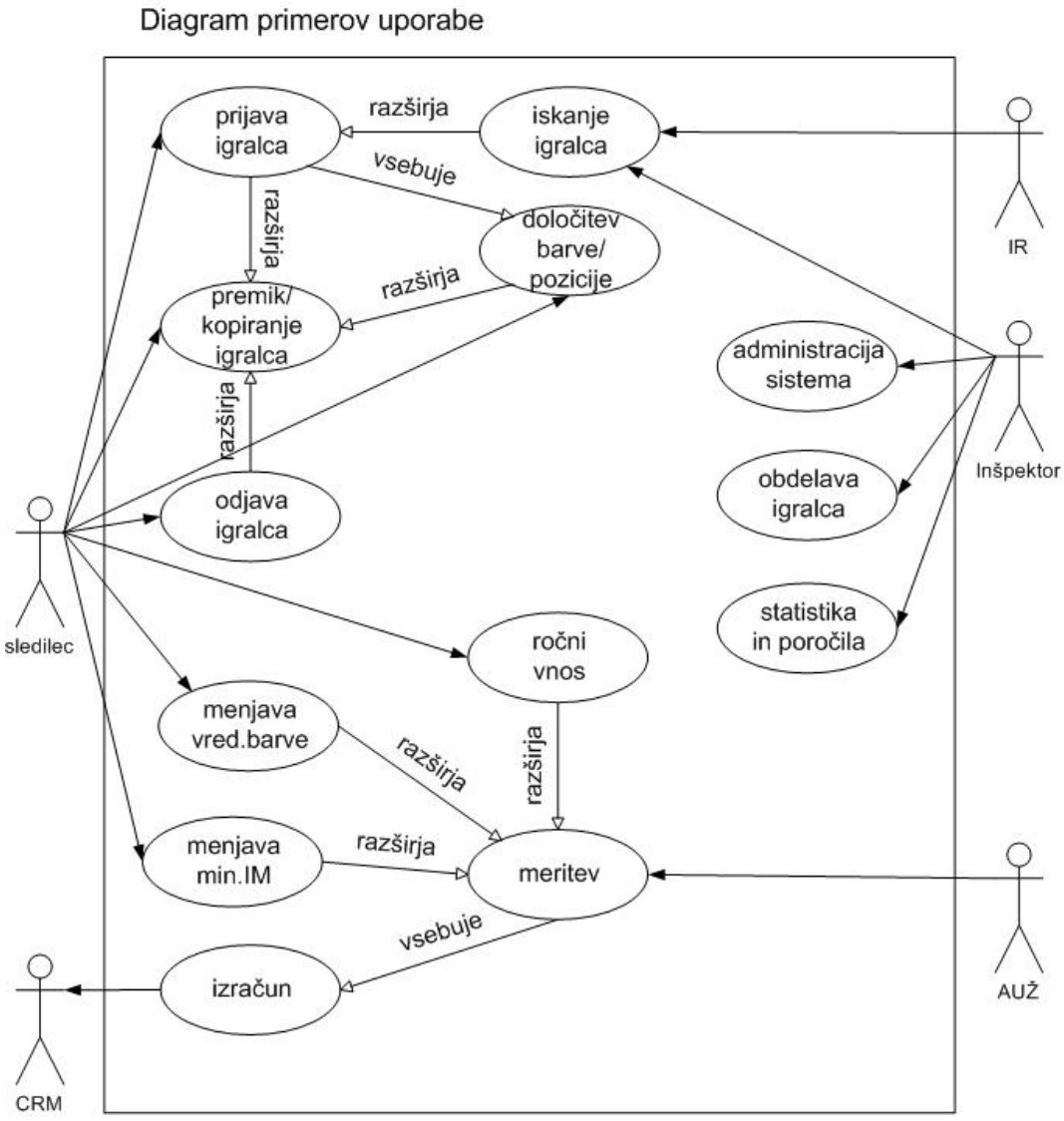
zapis meritve Ocena posamične igre gosta; gostu se vpiše dejanska ali povprečna stava oz. se avtomatično odčita stavo v barvnih žetonih, s katerimi igra.

3.2.2 Primeri uporabe

V tem podpoglavju je prikazan model primerov uporabe (slika 4). Podrobneje pa so opisani tisti primeri uporabe, ki vplivajo na izdelavo apliacije na mobilni napravi - dlančniku.

3.2.2.1 Prijava igralca

Kratek opis Za sledenje igri določenega igralca ga je potrebno najprej prijaviti v sistem. Če igralec na ameriški ruleti igra z barvnimi žetoni, je potrebno poleg njegove prijave dodati še barvo, s katero igra (tako lahko AUŽ avtomatično šteje njegovo igro). V primeru, da igralec igra z vrednostnimi žetoni, se njegovo igro lahko označi kot sledenje na podlagi dejanske ali povprečne stave. En igralec lahko ima na eni mizi več barv, lahko pa tudi igra na več mizah.



Slika 4: Model primerov uporabe

Osnovni tok

1. Akter označi polje `Ime in priimek` prazne vrstice (klik oz. dotik na polje).
2. Sistem izbrano polje `Ime in priimek` obarva v statusno barvo (npr. rumeno).
3. Sistem odpre novo okno - Iskalnik igralca.
4. Akter v iskalniku poišče in izbere igralca.
5. Sistem prenese v polje `Ime in priimek` izbranega igralca oz. NN-neznan igralec, če je bil ta izbran.
6. Sistem v desnem spodnjem statusnem oknu prikaže sliko izbranega igralca.
7. Sistem zabeleži dogodek izbire igralca.

Alternativni tok

- V koraku 3 sistem preveri, ali je v teku akcija `Kopiraj` ali `Prenesi` igralca. Če je akcija v teku, se izvede ta akcija namesto izbire igralca.

  Kopiranje igralca: funkcija prijavi igralca, ki je že aktivien ali neaktiven na katerikoli poziciji igralne mize na izbrano pozicijo. Pri tem ga pusti prijavljenega tudi na izvoru (na že prijavljeni poziciji).

  Prenos igralca: funkcija prijavi igralca, ki je že aktiven ali neaktiven na katerikoli poziciji igralne mize, na izbrano pozicijo. Pri tem ustavi meritev na izvoru in ga odjavi iz izvora (iz že prijavljene pozicije).

Predpogoji

- Akter je identificiran in avtoriziran.

3.2.2.2 Določitev barve

Kratek opis Po prijavi igralca v sistem mu je potrebno določiti:

- ameriška ruleta: Igralec lahko igra z barvnim žetonom ali z vrednostnim žetonom (sledenje po dejanski ali povprečni stavi). Izbira se lahko med barvnimi simboli, ki predstavljajo barvo žetonov (tako lahko AUŽ avtomatično šteje njegovo igro), ali simbolom denarja (igra z vrednostnimi žetoni - dejanska stava), ter simbolom roke (igra z vrednostnimi žetoni - povprečna stava).

- igre s kartami: izbrati je potrebno pozicijo, na kateri igra igralec (1A, 1B, 2A, 2B, ..., 6A, 6B).

Osnovni tok

1. Akter označi polje `Barva` v vrstici na kateri je določen igralec (klik oz. dotik na polje).
2. Sistem izbrano polje `Barva` obarva v statusno barvo (npr. rumeno).
3. Sistem odpre novo okno za izbiro barve.
4. Akter izbere barvo oz. dejansko ali povprežno stavo.
5. Sistem prenese v polje `Barva` izbran tip žetona, s katerim bo igralec igral (barvni, dejanska ali povprečna stava).
6. Sistem zabeleži dogodek povezave igralca z barvnim žetonom, dejansko ali povprečno stavo.
7. Sistem začne avtomatično šteti število žetonov, ki gredo skozi AUŽ (absolutni števci).

Alternativni tok

- Določitev pozicije igralca pri igrah s kartami
    1. Akter označi polje `Pozicija` v vrstici na kateri je določen igralec (klik oz. dotik na polje).
    2. Sistem izbrano polje `Pozicija` obarva v statusno barvo (npr. rumeno).
    3. Sistem odpre novo okno - Določitev pozicije.
    4. Akter izbere pozicijo, na kateri igra igralec.
    5. Sistem prenese v polje `Pozicija` izbrano pozicijo, na kateri bo igralec igral (1A, 1B, ..., 6A, 6B).
    6. Sistem zabeleži dogodek povezave igralca z pozicijo igre.
    7. Akter prične z ročnim sledenjem igre igralca (glej Napovedi vrednostne stave).

- Preklic izbire barve/pozicije
  V koraku 3 lahko akter prekliče izbiro barve/pozicije. Izbere gumb `Prekliči` in akcija se prekine.
  Ameriška ruleta: sistem ne prične šteti podatkov iz AUŽ, ker igralec nima določenega barvnega žetona. Onemogočen je ročen vnos stave za povprečno ali dejansko stavo.
  Igre s kartami: sistem ne omogoči ročnega sledenja igre igralca.

- Izbira dejanske ali povprečne stave
  V koraku 7 osnovnega toka sistem ne prične s štetjem žetonov, ki gredo skozi AUŽ. Akter bo ročno vnašal višino stave (akter izbere dejansko ali povprečno stavo, glej ročni vnos stave).

Predpogoji

- Akter je identificiran in avtoriziran.

3.2.2.3 Iskanje igralca

Kratek opis Ko se želi slediti igri določenega igralca, ga je potrebno prijaviti v sistem. Zaradi uvedbe "tihega sledenja" igre klubskih igralcev, je potrebno igralca poiskati v bazi podatkov in ga prijaviti v sistem. Iskalnik omogoča iskanje igralcev po določenih kriterijih.

Osnovni tok

1. Akter izbere filtre, ki služijo kot kriterij iskanja.

2. Akter izbere gumb `Najdi igralca`.

3. Sistem prikaže v novem oknu rezultat iskanja.

4. Akter označi izbranega igralca (s klikom oz. dotikom na sliko igralca, ki se obrobi z statusno barvo).

5. Potrdi izbiro igralca z gumbom `Izberi`.

6. Sistem prenese ime (ID) igralca v osnovno okno programa PRAUŽ na mesto, kjer se prijavlja igralca v sistem (glej Prijava igralca).

Alternativni tok

- Izberi neznanega igralca - NN
  V koraku 2 lahko akter izbere gumb `Neznan igralec (NN)`. Sistem prenese neznanega igralca (NN) v osnovno okno programa PRAUŽ na mesto, kjer se prijavlja igralca v sistem (glej Prijava igralca).
- Pregled predhodnih/nadalnjih zadetkov iskanja V koraku 4 lahko akter z izbiro gumba `Predhodnih 8` ali `Nadalnjih 8` pregleda predhodnih 8 ali nadalnjih 8 zadetkov iskanja po izbraneih kriterijih.
- Prekinitev izbire igralca
  V koraku 2 lahko akter izbere gumb `Prekliči`.

Predpogoji

- Akter je identificiran in avtoriziran.
- Akter je že izbral prijavo novega igralca na igralno mizo (glej Prijava igralca)
- Sistem odpre novo okno - Iskalnik igralca.

3.2.2.4 Premik / kopiranje igralca

Kratek opis Nad prijavljenim igralcem (tudi če še nima določene barve žetona ali pozicije) lahko izvedemo akcije premik ter kopiranje igralca.

Osnovni tok

1. Akter označi polno vrstico (klik oz. dotik na polje `Ime in priimek`).
2. Sistem izbrano vrstico obarva v statusno barvo (npr. rumeno).
3. Akter izbere gumb `Kopiraj igralca`.
4. Akter izbere ciljno pozicijo (lahko je ista ali druga igralna miza).
5. Akter izbere polje `Ime in priimek` prazne vrstice, kamor želi kopirati igralca.
6. Sistem v izbrano polje `Ime in priimek` vpiše igralca, ki je bil kopiran.
7. Sistem v desnem spodnjem statusnem oknu prikaže sliko izbranega igralca.
8. Sistem zabeleži dogodek kopiranja igralca.

Alternativni tok

- Izbira gumba `Prenesi igralca`
  V koraku 3 lahko akter izbere gumb `Prenesi igralca`. V tem primeru se ostali koraki izvajajo enako, le da se na koncu izvede še korak, v katerem sistem igralca odjavi na izvornem mestu in prekine njegovo štetje žetonov (zaključi njegovo meritev), ter si odjavo igralca zabeleži.

Predpogoji

- Akter je identificiran in avtoriziran.

3.2.2.5 Odjava igralca

Kratek opis Nad prijavljenim igralcem, ki ima določeno barvo žetona ali pozicijo lahko izvedemo akciji zaključka in odpovedi meritve.

Osnovni tok

1. Akter označi polno vrstico (klik oz. dotik na polje `Ime in priimek`).
2. Sistem izbrano vrstico obarva v statusno barvo (npr. rumeno).
3. Akter izbere gumb `Stop meritev`.
4. Če je bil igralec ročno sleden (dejanska, povprečna stava pri ameriški ruleti ali pozicija pri igrah s kartami), sistem odpre pokrivno okno ocene odjavljenega igralca.
5. Sistem odjavi igralca.
6. Sistem zaključi s štetjem žetonov iz AUŽ za izbranega igralca.
7. Sistem izprazni polje `Ime in priimek`.
8. Sistem izprazni polje `Barva` in s tem sprosti barvo.
9. Sistem izprazni polje `Čas sledenja`.
10. Sistem izprazni polje `Stava`.
11. Sistem zabeleži dogodek odjave igralca.
12. Sistem zabeleži meritev igre igralca (na podlagi preštetih žetonov ali vpisanih dejanskih ali povprečnih stav).

Alternativni tok

- Izbira gumba `Zavrži meritev`
  V koraku 3 lahko akter izbere gumb `Zavrži`. Akcija zavrže vse meritve izbranega igralca in sistem ga odjavi iz primarne pozicije.

    4. Sistem zahteva potrditev akcije.
    5. Sistem odjavi igralca na izvornem mestu in prekine štetje žetonov (zaključi njegovo meritev). Pri tem na izvornem mestu izprazni polja `Ime in priimek`, `Barva/Pozicija`, in `Stava`.
    6. Sistem zabeleži dogodek preklic meritve igralca.

Predpogoji

- Akter je identificiran in avtoriziran.

3.2.2.6 Ročni vnos stave

Kratek opis Prijavljenemu igralcu, ki ima določeno dejansko ali povprečno stavno pri ameriški ruleti ali pozicijo pri igrah s kartami, lahko vnašamo stave, na podlagi katerih se izračuna meritev.

Osnovni tok

1. Akter označi polje `Stava` (klik oz. dotik na polje).
2. Sistem izbrano polje `Stava` obarva v statusno barvo (npr. rumeno).
3. Sistem odpre novo prekrivno okno za vnos stave.
4. Akter vnese vrednost stave (s hitrimi gumbi ali točnim vnosom stave).
5. Sistem zabeleži vnešeno stavo
6. Sistem v polje `Stava` vpiše vrednost vnešene stave.

Alternativni tok

- Vpis visoke povprečne stave
  Ko se vpiše stavo, se ob previsokem vpisu stave (n-krat večja vrednost od zadnje vpisane stave) prikaže obvestilo o previsokem vnosu.
  V koraku 5 sistem zahteva potrditev visoke vpisane vrednostne stave.

  6. Akter lahko prekliče pevisok vnos (gumb `Prekliči`) ali ga potrdi (gumb `Dodaj`).

- Preklic izbire vrednosti stave
  V koraku 4 lahko akter prekliče izbiro vpisa stave.

  5. akter izbere gumb "Prekliči" in akcija se prekine.

Predpogoji

- Akter je identificiran in avtoriziran.

3.3 Analiza in načrtovanje

3.3.1 Podatkovni model

3.3.1.1 Opis podatkovnega modela

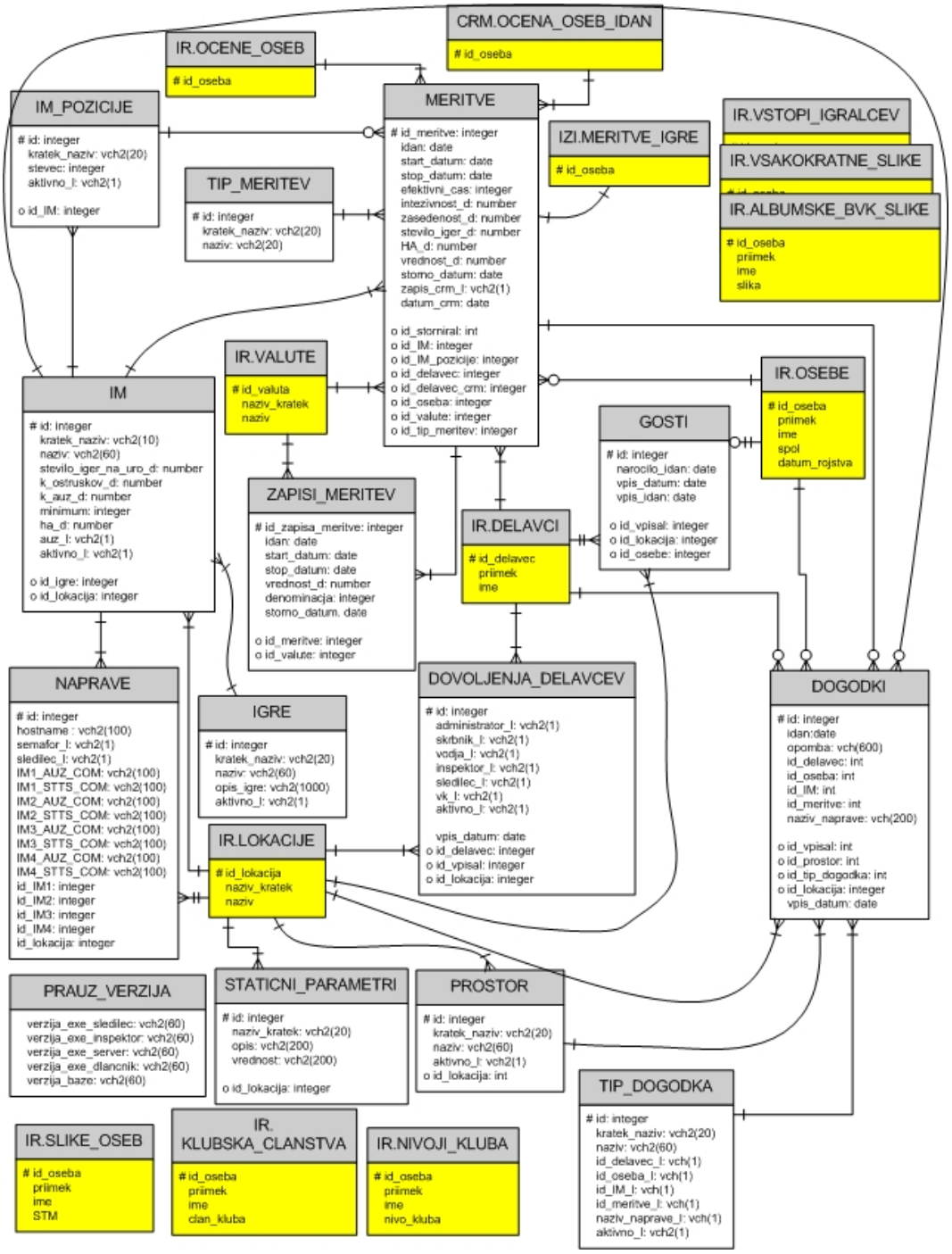
IR.DELAVCI Tabela je definirana v shemi IR. Vsebuje delavce, ki bodo sledili igralcem v PRAUŽ. Atributi so `ID delavca, Ime, Priimek`.

IR.OSEBE, IR.SLIKE_OSEB Tabeli sta definirani v shemi IR. Vsebujejta podatke o igralcih, ki bodo sledeni v PRAUŽ. Atributi so `ID Igralca, Ime, Priimek, Spol, Rojstni datum` in `slika osebe`.

IR.VSAKOKRATNE_SLIKE Tabela je definirana v shemi IR. Vsebuje slike zajete ob vsakem vstopu v igralnico.

IR.ALBUMSKE_BVK_SLIKE Tabela je definirana v shemi IR. Vsebuje kvalitetnejše slike, zajete ob prvem obisku igralnice.

IR.KLUBSKA_CLANSTVA, IR.NIVOJI_KLUBA Tabeli sta definirani v shemi IR. Vsebujeta podatke o članstvu v klubih. Atribut je naziv nivoja klubske kartice (npr.: zlata, srebrna kartica).



Slika 5: Podatkovni model

IR.LOKACIJE Tabela je definirana v shemi IR. Vsebuje igralnice, ki jim pripadajo entitete[3] v PRAUŽ. Atributi so `ID Lokacije, kratek naziv` in `naziv`.

IR.OCENE_OSEB PRAUŽ vsako zaključeno meritev vpisuje v to tabelo.

CRM.OCENA_OSEB_IDAN PRAUŽ vsako zaključeno meritev vpisuje tudi v to tabelo.

IR.VSTOPI_IGRALCEV Tabeli sta definirani v shemi IR. Vsebujeta podatke o vstopih v igralnico. Ključ je datum vstopa v igralnico.

IR.VALUTA Tabela je definirana v shemi IR. Vsebuje valute, v katerih so možna vplačila.

IM_POZICIJE Tabela določa AUŽ napravo. Z njo opišemo, katere pozicije so aktivne na igralni mizi oziroma katero barve so aktivne v AUŽ na ameriški ruleti.

TIP_MERITEV Tabela določa tipe meritev. Ko začnemo sledenje igralcu, moramo določiti tip meritve.

MERITVE Zaključek spremljanja igre igralca se zabeleži v eno meritev. Na igralni dan lahko ima igralec zabeleženih več meritev. Skupek vseh njegovih meritev je osnova za oceno igre igralca na igralni dan.

ZAPISI_MERITEV Tabela stav igralca. Pri igrah s kartami ter pri povprečni in dejanski stavi na ameriški ruletu je vsak vpis stave shranjen v svojem zapisu v tabeli. Pri sledenju na ameriški ruleti z avtomatskim urejevanikom žetonov pa je en zapis za vse meritve pri eni denominaciji (vrednosti) barvnih žetonov.

IM Tabela igralnih miz, na katerih se lahko izvaja sledenje igre igralcev.

[3]Entitete so objekti (osebki) iz realnega sveta o katerih zbiramo, obdelujemo in hranimo podatke in informacije. Primeri entitet: človek Miha Novak, knjiga Vojna in mir, avto Ford Fiesta, ...

GOST Tabela gostov (igralcev), ki igrajo na igralnih mizah in so zanimivi za sledenje. Če gost ni več aktualen, ga pobrišemo iz tabele.

NAPRAVE Tabela naprav, ki se uporabljajo pri sledenju (npr. avtomatski urejevanik žetonov)

IGRE Tabela vseh iger, ki se lahko odvijajo na igralnih mizah.

DOVOLJENJA_DELAVCEV Tabela pravic delavcev določa, katere akcije lahko delavec aktivira znotraj aplikacije.

DOGODKI Tabela zaznamkov pomembnejših dogodkov, ki so povezani s sledenjem iger na igralnih mizah.

TIP_DOGODKA Tabela tipov dogodkov, ki definirajo, s čim je dogodek povezan (naprava, igralec, zaposlen, ...).

PROSTOR Tabela prostorov igralnice, kjer se je zgodil posamezen dogodek.

STATIČNI_PARAMETRI Tabela vseh statičnih parametrov, ki so potrebni za delovanje aplikacije.

PRAUZ_VERZIJA Tabela z podatki o zadnji verziji posameznega modula aplikacije.

3.4 Arhitektura aplikacije

Aplikacija je zasnovana tronivojsko. Na prvem nivoju se nahaja podatkovni del - podatkovna baza z tabelami, relacijami med njimi ter ostalimi podatki. V našem primeru je uporabljena podatkovna baza Oracle.

Na drugem nivoju se nahajata plast poslovne logike in plast za dostop do podatkov. Plast poslovne logike ima funkcijo izvajanja poslovnih pravil, plast za dostop do podatkov pa pripravi podatke v obliki za prikaz in shranjevanje. Ta nivo je narejen kot aplikacijski strežnik, na katerem so realizirane vse akcije in z njimi povezana poslovna pravila, ki so navedene v poglavju zajema zahtev.

Tretji nivo je predstavitveni nivo - grafični uporabniški vmesnik. Ta prikazuje podatke pridobljene s aplikacijskega strežnika ter proži akcije na njem. Za komunikacijo med predstavitvenim in poslovnim nivojem je uporabljena tehnologija .NET Remoting.

3.5 Predstavitev modulov aplikacije

Aplikacija je zaradi različnih namenov uporabe razdeljena na dva modula: `Inšpektorski modul` in `Sledilec`.

3.5.1 Inšpektorski modul

Inšpektorski modul je namenjen za nadzor, pregled poročil in administracijo. Nameščen je na nekaterih računalnikih v igralnici in v pisarnah inšpektorjev.

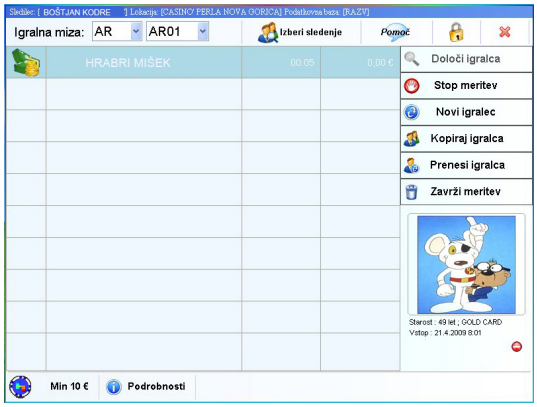
3.5.2 Sledilec

Modul `Sledilec` je namenjen sledenju igralcem na igralnih mizah. Pri tem modulu je posebnost to, da mora delovati na dveh operacijskih sistemih (Microsoft Windows in Linux), ker je v različnih igralnicah nameščen različen operacijski sistem. Zaradi te zahteve je bilo potrebno pri razvoju uporabljati le tiste komponente .NET ogrodja, ki so podprte tudi v Mono.

Uporaba aplikacije se začne z prijavo ob zagonu. Prijava je narejena tako, da preveri če ima računalnik nameščen RFID čitalec. Če ga ima, se od uporabnika zahteva prijava z RFID kartico, v nasprotnem primeru pa se prijavi z vnosom številčnega gesla (slika 6). Ob prijavi se preverijo tudi pravice uporabnika, na podlagi katerih je dovoljen vstop v modul `Sledilec`. Ob uspešni prijavi se nam pokaže osnovno okno `Sledilca` (slika 7). To okno ima lahko prednastavljeno igralno mizo (na računalnikih posameznih igralnih miz) ali pa omogoča najprej izbor igre, nato pa glede na izbrano igro še izbor mize. Ko imamo izbrano mizo lahko začnemo slediti igralcem na tej mizi. Sledenje začnemo z določitvjo igralca (slika 8), ki ga lahko poiščemo z:

- vnosom ID-ja igralca,
- vnosom imena in priimka igralca in
- z izborom starostne lestvice (do 30 let, 20 - 40, 30 - 50, ...)

Slika 6: Prijava v modul Sledilec



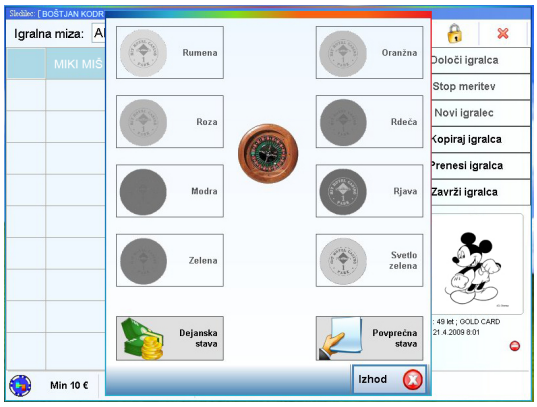
Slika 7: Sledilec z enim aktivnim sledenjem

Slika 8: Iskanje gosta

Pri vseh iskanjih (razen po ID-ju) se nam prikaže okno z slikami igralcev (slika 9), ki ustrezajo izbranim kriterijem. Na podlagi teh slik se nato izbere željenega igralca.

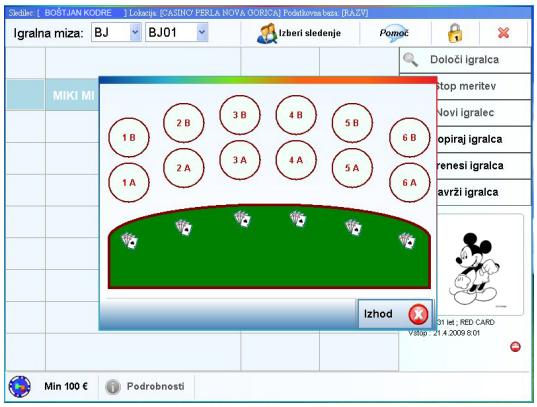
Slika 9: Prikaz gostov, ki ustrezajo iskalnim kriterijem

Naslednji korak se razlikuje glede na izbrano igro. Tako je pri ameriški ruleti ta korak izbira načina sledenja igralcu (slika 10). Izbiramo lahko med naslednjimi

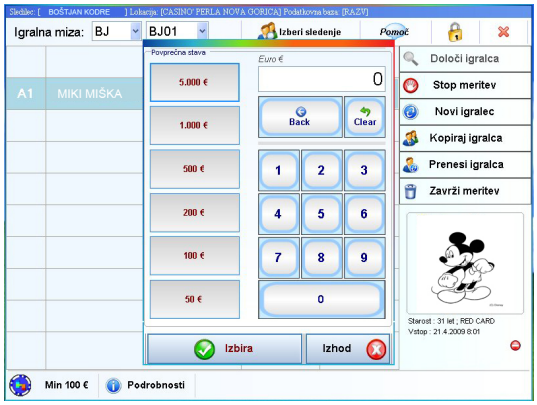


Slika 10: Izbor pozicije pri ruleti

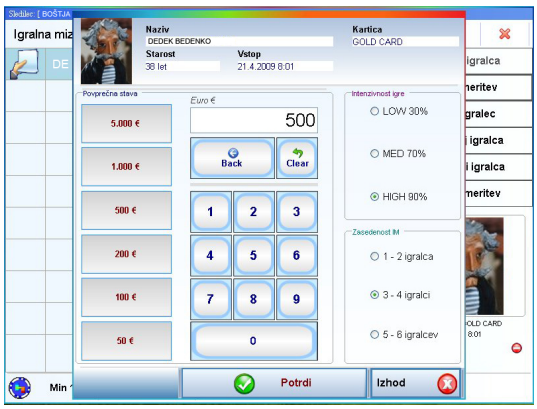
tipi sledenj: glede na povprečno, dejansko stavo, ali določitvjo barve žetona. Pri določitvi barve žetona nam avtomatski urejevalnik žetonov vrača število zaigranih žetonov, na podlagi katerih se izračunajo stave. Pri igrah s kartami namesto tipa sledenja izbiramo pozicijo igralca (slika 11). Igralcu z določeno pozicijo ali tipom sledenja lahko dodajamo stave (slika 12), zavržemo meritev, zaključimo meritev (slika 13), prenašamo ali kopiramo igralca na drugo mizo.



Slika 11: Izbor pozicije pri igrah s kartami



Slika 12: Vnos stave



Slika 13: Zaključevanje sledenja

Poglavje 4

Izdelava mobilne aplikacije

4.1 Problemska domena

4.1.1 Opis problema

PRAUŽ je bil narejen za sledenje igralcem ter pridobivanje čim točnejših podatkov o stavah. Vendar so se pri uporabi PRAUŽ-a pojavljali logistični problemi, kot so:

- pomanjkanje računalnikov na igralnih mizah pri igrah s kartami; posledica je pisanje stav igralcev na list papirja ter naknadno vnašanje v PRAUŽ

- pri sledenju preko papirja je težavno določanje igralca; v PRAUŽ-evem modulu `Sledilec` to naredimo s pomočjo slik vstopa igralcev v igralnico

- težavnost sledenja več igralcem; sledilec zelo težko sledi stavam več igralcev, kar lahko vodi do slabih podatkov. Alternativa je sledenje manjšemu številu igralcev.

Iz teh težav je razvidna potreba po mobilnosti sledilcev, ki ne bi smeli uporabljati zamudnega papirnega sledenja, temveč bi izvajali sledenje s pomočjo mobilne aplikacije. S tem bi znatno prihranili na času sledenja, saj ne bi bilo več potrebe po naknadnem vpisovanju v sistem, sledilec pa lahko z mobilno aplikacijo sledi tudi več igralcem. Tako bi se lahko sledilci premikali med mizami in sledili več igralcem na bližnjih mizah naenkrat, lahko pa bi več sledilcev tudi sledilo igralcem na eni mizi.

4.1.2 Glavne zahteve

Iz zgoraj opisanih problemov pridemo do sledečih zahtev:

- določanje gostov na podlagi slik vstopa v igralnico,
- vpisovanje stav igralcev in
- zaključevanje sledenja (meritev)

4.2 Razvoj aplikacije

Ker je ta aplikacija le verzija PRAUŽ-evega modula `Sledilec` za dlančnik, nismo imeli potrebe po zajemu zahtev, analizi in načrtovanju, saj smo imeli to narejeno pri namizni verziji aplikacije. To, da je PRAUŽ narejen tronivojsko nam je še dodatno olajšalo razvoj, ker smo imeli narejeno večino poslovne logike na strežniku in je bilo potrebno razviti le predstavitveni nivo ter zagotoviti ustrezno komunikacijo z strežnikom. Pri razvoju smo uporabili evolucijski pristop, razlog za to pa so bile predvsem nejasnosti, kako realizirati sledenje na mizah pri ameriški ruleti. Tu modul `Sledilec` podpira tri različne vrste sledenj glede na povprečno stavo, dejansko stavo, ter glede na barvo žetona (stava se izračuna glede na števec žetonov izbrane barve na avtomatskem urejevalniku žetonov). Težavni sta zlasti zadnji dve: dejanska stava zaradi možnosti podvajanja vnosa stave v `Sledilcu` ter mobilnemu `Sledilcu`, sledenje glede na barvo pa zaradi tega, ker na dlančniku ne moremo pridobiti podatkov o števcih žetonov iz avtomatskega urejevalnika žetonov. Tako je v prvi iteraciji razvoja podprto le sledenje na mizah pri igrah s kartami, ostale funkcionalnosti pa se je dodajalo v naslednjih iteracijah, glede na povratne informacije uporabnikov. Postopni razvoj, spreminjanje ter dodajanje funkcionalnosti glede na povratne informacije uporabnikov so tipične značilnosti evolucijskega razvoja.

4.3 Grafični uporabniški vmesnik

Grafični uporabniški vmesnik ima zelo velik pomen v aplikaciji, saj je vmesni člen med uporabnikom in aplikacijo. Zato je pomembno da je grafični uporabniški vmesnik:

- intuitiven in enostaven; uporabniki tako potrebujejo manj usposabljanj in podpore za uporabo, večja je motiviranost za delo.

- usklajen skozi celotno aplikacijo; Gumbi in meniji z istim pomenom so vedno na istem mestu, funkcijske tipke in dvojni klik imajo vedno isto funkcionalnost, itd.
- ima pravilno razvrščene gradnike; razvrstitev se mora skladati s potekom dela.
- prehodi med okni aplikacije so v skladu z potekom dela.
- gradniki so ustrezno grupirani; povezani gradniki naj bodo skupaj, nepovezani pa narazen. Posamezne grupe gradnikov ločujemo z barvami ali okvirji.

Poleg zgoraj naštetih lastnosti, ki so bolj funkcionalne narave, je potrebno poskrbeti tudi za estetski del, ki ga določamo z:

- barvami; Pri pravilni uporabi je uporabniški vmesnik bolj prijazen, poudarjeno je grupiranje informacij, opozorila bolj pritegnejo pozornost. Za uporabo so priporočene pastelne barve, saj žive barve dražijo uporabnika(recimo živo rdeča barva). Posebej je treba paziti na kontraste, ki lahko drastično vplivajo na berljivost.
- pisavami; priporočena je uporaba čimmanj različnih pisav znotraj ene aplikacije. Priporočena je tudi uporaba `sans serif` pisav zaradi lažje berljivosti.
- ikonami; z njimi ustrezno predstavimo objekte, akcije, ...

Pri izdelavi grafičnega vmesnika, smo se želeli čim bolj držati napisanih smernic, kar pa se je pri razvoju za .NET kompaktno ogrodje pokazalo kot precejšen iziv. V .NET kompaktnem ogrodju je veliko pomankanje vizualnih gradnikov, ki bi nam omogočali izdelavo željenega grafičnega vmesnika, pa tudi pri obstoječih gradnikih marsičesa ni mogoče nastaviti. Iz tega razloga se je pojavila potreba po izdelavi lastnih gradnikov, kot so: `group box, table panel, button, keyboard`.

4.3.1 Razvoj vizualnih gradnikov

Razvoj vizualnih gradnikov začnemo tako, da v Visual Studiu odpremo nov projekt za pametne naprave (ang. Smart device project) in za tip projekta izberemo knjižnjico razredov (ang. class library). V ta projekt nato dodajamo nove vizualne gradnike tako, da v menuju projekta izberemo dodaj

komponento (ang. `add component`). Komponento izberemo zato, ker nam ta za razliko od rezreda sama zgenerira metodi za inicializacijo komponente (`InitializeComponent`), ter za uničenje komponente (`Dispose`) - sprostitev zaseženega pomnilnika. V naslednjem koraku pravilno nastavimo dedovanje našega gradnika. Če želimo dodati funkcionalnost že obstoječemu gradniku, lahko dedujemo od obstoječega gradnika, če pa želimo narediti nov gradnik, dedujemo od gradnika `System.Windows.Forms.Control`. V teku razvoja aplikacije smo razvili več vizualnih gradnikov, zato se bomo za prikaz razvoja osredotočili na izdelavo enega samega gradnika - v našem primeru gumba. Pri izdelavi gumba se nismo želeli omejevati z obstoječim gumbom, zato smo dedovali od gradnika `Control`. Nato smo dodali željene lastnosti gumba: pisava, tekst, barva teksta, poravnava teksta, slika, razmerje med tekstom in sliko, začetna barva ozadja, končna barva ozadja, tip ozadja, tip gradienta, debelina roba, barva roba, barva pritisnjenega gumba. Nekatere lastnosti (kot so pisava, tekst, barva teksta) že obstajajo v gradniku `Control` in nam jih zato ni bilo treba dodajati. Pri vseh lastnostih, ki določajo izgled, je potrebno po nastavljanju nove vrednosti poskrbeti za ponoven izris gumba. To dosežemo z ukazi `Refresh()` ali `Invalidate()`. Dodajanju lastnosti sledi izdelava izgleda. Tega se lotimo tako, da povozimo metodi `OnPaintBackground` in `OnPaint`. V metodi `OnPaintBackground` naredimo izris ozadja našega gradnika glede na stanje gumba. Stanja gumba za .NET kompaktno ogrodje so: omogočen, onemogočen ter pritisnjen (gumb v .NET ogrodju ima še četrto stanje, ko je kurzor nad gumbom). Za onemogočen ter pritisnjen gumb smo si zamislili enobarvno ozadje, iz česar sledi naslednja koda:

```
Brush brush = new SolidBrush(mouseDownBackColor);
e.Graphics.FillRectangle(brush, e.ClipRectangle);
brush.Dispose();
```

Pri omogočenem gumbu pa smo želeli lepši izris, zato smo dodali izbiro za tip ozadja, ki je lahko enojna barva ali pa gradientni prehod. Ob izboru enojne barve je metoda za izris enaka onemogočenemu ter pritisnjenemu gumbu, ob izboru gradientnega prehoda pa je izris odviesen še od izbire tipa gradienta (`Horizontal, Verical, FromMiddleHorizontal, FromMiddleVertical`):

```
if (GradientType == GradientType.Horizontal)
{
  LinearGradientBrushEx brLinGrad;
  brLinGrad = new LinearGradientBrushEx(rc,
      backColorStart, backColorEnd,
```

```
      DrawingExtended.LinearGradientMode.Horizontal);
  g.FillRectangle(brLinGrad, rc);
  brLinGrad.Dispose();
}
else if (GradientType == GradientType.Vertical)
{
  LinearGradientBrushEx brLinGrad;
  brLinGrad = new LinearGradientBrushEx(rc,
      backColorStart, backColorEnd,
      DrawingExtended.LinearGradientMode.Vertical);
  g.FillRectangle(brLinGrad, rc);
  brLinGrad.Dispose();
}
else if (gradientType ==
    GradientType.FromMiddleHorizontal)
{
  LinearGradientBrushEx brLinGrad;
  brLinGrad = new LinearGradientBrushEx(rc,
      backColorStart, backColorEnd,
      DrawingExtended.LinearGradientMode.Horizontal);
  ColorBlendEx cb = new ColorBlendEx(3);
  cb.Colors[0] = backColorStart;
  cb.Colors[1] = backColorEnd;
  cb.Colors[2] = backColorStart;
  cb.Positions[0] = 0;
  cb.Positions[1] = 0.5f;
  cb.Positions[2] = 1.0f;
  brLinGrad.InterpolationColors = cb;
  g.FillRectangle(brLinGrad, rc);
  brLinGrad.Dispose();
}
else if (gradientType ==
    GradientType.FromMiddleVertical)
{
  LinearGradientBrushEx brLinGrad;
  brLinGrad = new LinearGradientBrushEx(rc,
      backColorStart, backColorEnd,
      DrawingExtended.LinearGradientMode.Vertical);
  ColorBlendEx cb = new ColorBlendEx(3);
```

```
  cb.Colors[0] = backColorStart;
  cb.Colors[1] = backColorEnd;
  cb.Colors[2] = backColorStart;
  cb.Positions[0] = 0;
  cb.Positions[1] = 0.5f;
  cb.Positions[2] = 1.0f;
  brLinGrad.InterpolationColors = cb;
  g.FillRectangle(brLinGrad, rc);
  brLinGrad.Dispose();
}
```

Metoda `OnPaint` je namenjena izrisu ostalih vizualnih lastnosti, v našem primeru roba gumba, teksta ter slike. Če ima gumb nastavljeno sliko in tekst, moramo izračunati njuno pozicijo glede na razmerje med njima (`ImageBeforeText`, `ImageAboveText`, `TextBeforeImage`, `TextAboveImage`, `Overlay`), ter glede na njuno poravnavo (`TopLeft`, `TopCenter`, `TopRight`, `MiddleLeft`, `MiddleCenter`, `MiddleRight`, `BottomLeft`, `BottomCenter`, `BottomRight`). Podobno kot pri izrisu ozadja gumba je tudi tu potrebno narediti izris za več stanj gumba; omogočen ter onemogočen. Pri izrisu onemogočenega gumba je potrebno posiviti sliko na gumbu. Ker so operacije nad slikami strojno zahtevne, smo v našem primeru delno posivitev dosegli z izrisom delno prosojnega sivega pravokotnika čez sliko:

```
if (!Enabled)
{
  Color c= Color.FromArgb(0x30C2C7C6);
  g.FillRectangle(new SolidBrushEx(c), x, y,
      image.Width, image.Height);
}
```

Da dosežemo spremembo barve ozadja ob kliku na gumb je potrebno redefinirati metodi `OnMouseDown` in `OnMouseUp`. V metodi `OnMouseDown` nastavimo spremenljivko `mouseDown` na `true` in sprožimo `Invalidate()`, v `OnMouseUp` pa nastavimo `mouseDown` nazaj na `false` in ponovno sprožimo `Invalidate()`. Ob klicu `Invalidate()` se gumb ponovno izriše glede na nastavljeno stanje(določata ga spremenljivki `mouseDown` in `enabled`). Gradnik `Control` ima veliko dogodkov, eden izmed njih je tudi dvojni klik (`doubleclick`), ki pa ga pri gumbu ne potrebujemo in nam kvari normalno delovanje gumba. Za rešitev te problematike je potrebno redefinirati metodo `OnDoubleClick` in v njej sprožiti dogodek `OnClick`, da dobimo željeno delovanje. Ker nočemo, da

ima naš gumb dogodek za dvojni klik, odpremo razredni diagram (ang. Class Diagram) projekta, izberemo razred `Gumb` ter mu v oknu lastnosti dodamo atribut `DesktopCompatible(true)`. Nato se nam v projektu pojavi datoteka `DesignTimeAttributes.xmta` v katero dodamo pod razred gumb:

```
<Event Name="DoubleClick">
  <Browsable>true</Browsable>
  <EditorBrowsable>true</EditorBrowsable>
</Event>
```

Po enakem postopku kot dvojni klik lahko skrijemo še ostale (pri gumbu neuporabne) dogodke in gumb je dokončno primeren za uporabo. Podobno kot gumb lahko izdelamo tudi druge gradnike, ker lahko nekateri znotraj sebe vsebujejo zbirke gradnikov (recimo: `TablePanel` vsebuje vrstice in stolpce, vrstica pa nato glede na število stolpcev še celice), kar znatno poveča kompleksnost gradnika.

4.3.2 Razvoj knjižnice za risanje

V teku razvoja vizualnih gradnikov se je pokazalo, da knjižnica za risanje v .NET kompaktnem ogrodju ne podpira vseh funkcij, ki bi nam omogočale izdelavo zastavljenih vizualnih gradnikov. Temu je sledilo preučevanje možnosti, kako bi izdelali manjkajoče funkcije z uporabo že obstoječih. Tako smo izdelali funkcije za izris kotov in gradientnega prelivanja, zalomilo pa se je pri izrisu slik. Pri slikah je težava v tem, da .NET kompaktno ogrodje ter Windows Mobile do verzije 5 podpira samo RGB paleto, zaradi česar vse slike z `alfa kanalom` izriše napačno. Po temeljitem raziskovanju problema smo ugotovili, da ima Windows Mobile neuradno podprto GDI+ knjižnico za risanje. Glede na to, da je knjižnica za risanje v .NET kompaktnem ogrodju ovoj knjižnice GDI, v .NET ogrodju pa ovoj GDI+, je postal razvoj ovoja GDI+ za .NET kompaktno ogrodje smiselna odločitev. Prvi problem, na katerega smo naleteli pri izdelavi ovoja je bilo pomankanje dokumentacije za GDI+ v Windows Mobile, zato smo si pomagali z dokumentacijo GDI+ za Windows Xp. Izdelave ovoja smo se lotili tako, da smo postavili pravila strukturiranja kode v projektu. Določili smo, da se zaradi preglednosti razred z zunanjimi funkcijami razbije na več datotek glede na objekt, na katerega se funkcije nanašajo (`Bitmap, Brushes, Graphics, Pen,` ...). Te datoteke se hranijo v mapi `NativeMethods`. Razrede, ki ne vsebujejo zunanjih funkcij, pa hranimo v korenu projekta.

4.3.2.1 Razred z zunanjimi funkcijami

Izdelavo razreda z zunanjimi funkcijami[1] smo si zastavili tako, da postopno dodajamo funkcije glede na pogostost uporabe. Tak način smo izbrali zato, ker dodajamo funkcije glede na dokumentacijo GDI+ za Windows Xp in posledično pričakujemo, da niso vse v dokumentaciji navedene funkcije podprte tudi na Windows Mobile. GDI+ je potrebno pred uporabo inicializirati, po uporabi pa sprostiti zasežene vire, zato smo najprej podprli ti dve funkciji. Za inicializacijo je potrebno klicati zunanjo funkcijo, katere deklaracija v `C++` programskem jeziku izgleda takole:

```
Status GdiplusStartup(
    ULONG_PTR token *token,
    const GdiplusStartupInput *input,
    GdiplusStartupOutput *output
);
```

Da bomo lahko naredili ekvivalenten klic v C# , moramo najprej definirati tipe kot so `Status, GdiplusStartupInput` in `GdiplusStartupOutput`. Status je naštevalni tip, pri katerem posamezna številka določa vrnjen status (`Ok = 0`, `GenericError = 1`, `InvalidParameter = 2`, `OutOfMemory = 3`, ...). `GdiplusStartupInput` je kompleksnejši tip, katerega definicija v C++ je:

```
struct GdiplusStartupInput
{
    UINT32 GdiplusVersion;
    DebugEventProc DebugEventCallback;
    BOOL SuppressBackgroundThread;
    BOOL SuppressExternalCodecs;

    GdiplusStartupInput(
        DebugEventProc debugEventCallback = NULL,
        BOOL suppressBackgroundThread = FALSE,
        BOOL suppressExternalCodecs = FALSE)
    {
        GdiplusVersion = 1;
        DebugEventCallback = debugEventCallback;
        SuppressBackgroundThread =
            suppressBackgroundThread;
```

[1] imenuje se `GDIPlus`

```
        SuppressExternalCodecs = suppressExternalCodecs;
    }
};

typedef VOID (WINAPI *DebugEventProc)(DebugEventLevel
    level, CHAR *message);
```

Pri pretvorbi kode v C# tip `UINT32` nadomestimo z `UInt32`, ter `BOOL` z `bool`. `DebugEventProc` je kazalec na metodo, to bi v C# lahko naredili z delegatom, ki je prav tako nekakšen kazalec na metodo. Ker pa mi te metode ne potrebujemo, smo kodo poenostavili in smo `DebugEventProc` zamenjali z tipom `IntPtr`. Po pretvorbi bi bil `GdiplusStartupInput` takšen:

```
[StructLayout(LayoutKind.Sequential)]
    internal struct GdiplusStartupInput
    {
        internal UInt32 GdiplusVersion;
        internal IntPtr DebugEventCallback;
        internal bool SuppressBackgroundThread;
        internal bool SuppressExternalCodecs;

        internal GdiplusStartupInput(
            IntPtr  debugEventCallback,
            bool suppressBackgroundThread,
            bool suppressExternalCodecs)
        {
            GdiplusVersion = 1;
            DebugEventCallback = debugEventCallback;
            SuppressBackgroundThread =
                suppressBackgroundThread;
            SuppressExternalCodecs =
                suppressExternalCodecs;
        }

        internal static GdiplusStartupInput
            CreateGdiplusStartupInput()
        {
            GdiplusStartupInput result =
                new GdiplusStartupInput();
            result.GdiplusVersion = 1;
```

```
            result.DebugEventCallback = IntPtr.Zero;
            result.SuppressBackgroundThread = false;
            result.SuppressExternalCodecs = false;
            return result;
        }
    }
```

Po istem principu naredimo tudi tip `GdiplusStartupOutput`. Ko imamo narejene vse potrebne tipe, lahko pretvorimo še deklaracijo metode `GdiplusStartup`. Za deklaracijo zunanje metode je potrebno v C# navesti `dll`, v katerem se nahaja metoda, ter označiti, da gre za zunanjo metodo. `Dll` navedemo tako, da dodamo metodi atribut `[DllImport("gdiplus.dll")]`, označbo zunanje metode pa izvedemo z modifikatorjem `extern`. Parametri metode `GdiplusStartup` so kazalci na objekt, kar v C# dosežemo tako, da pred parameter dodamo `ref`. Na ta način smo dobili sledečo deklaracijo metode:

```
[DllImport("gdiplus.dll")]
extern public static Status GdiplusStartup(
    ref ulong token,
    ref GdiplusStartupInput input,
    ref GdiplusStartupOutput output);
```

Da bi poskrbeli za avtomatsko inicializacijo ter sproščanje GDI+ virov, naredimo v našem razredu statični konstruktor, ki inicializira gdi+, ter ob koncu procesa sprosti zasežene vire:

```
static GDIPlus()
{
    GdiplusStartupInput input =
        GdiplusStartupInput.CreateGdiplusStartupInput();
    GdiplusStartupOutput output =
        new GdiplusStartupOutput();
    GdiplusStartup(ref GDIPlusToken,ref input,
        ref output);
    System.Diagnostics.Process p=
        System.Diagnostics.Process.GetCurrentProcess();
    p.Exited += new EventHandler(GDIPlus_Exited);
}

static void GDIPlus_Exited(object sender, EventArgs e)
{
```

```
    GC.Collect();
    GC.WaitForPendingFinalizers();
    GdiplusShutdown(GDIPlusToken);
}
```

Na enak način kot smo deklarirali metodo `GdiplusStartup` smo izdelali tudi deklaracije drugih zunanjih metod.

4.3.2.2 Izdelava knjižnice

Knjižnica `DrawingExtended` je ovoj GDI+, s katero ta komunicira preko razreda `GDIPlus`. Pri izdelavi `DrawingExtended` merimo na to, da bo imela podobne funkcionalnosti kot knjižnica `Drawing` v .NET ogrodju. Zaradi tega smo se tudi pri strukturiranosti `DrawingExtended` zgledovali po njej in smo imena razredov poimenovali približno enako. Tako je `Graphics` postal `GraphicsEx`, `Brush` je postal `BrushEx`, itd. Nekaterih razredov v `DrawingExtended` nismo naredili, ker potrebni klici v GDI+(posledično tudi v GDIPlus) niso implementirani. Taki so vsi rezredi povezani s pisavami `Font, FontFamily,` itd. Vsi razredi znotraj `DrawingExtended`, ki opisujejo objekt iz GDI+, imajo enako zasnovo. Pod enako zasnovo štejemo, da imajo:

- kazalec na izvorni objekt, ki ga potrebujemo pri vseh klicih, ki se nanašajo na ta objekt,
- konstruktor, v katerem se ta izvorni objekt inicializira,
- razne metode za delo z objektom in
- destruktor (metoda `dispose`), ki skrbi za sproščanje zaseženega pomnilnika. Pri implementaciji te metode je potrebna velika pozornost, da sprostimo vse zasežene vire s strani izvornega objekta, ker bodo drugače ostali ti viri nesproščeni. Smetar namreč ne more sproščati pomnilnika, ki ni zasežen s strani CLR-ja.

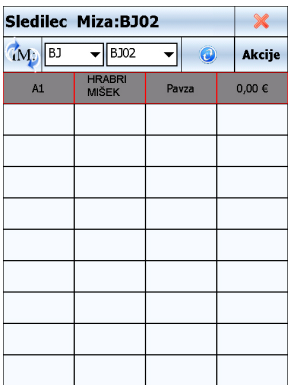
Pri metodah za delo z objekti gre predvsem za to, da kličemo prave metode `GDIPlus-a`. Ker tudi tu mankajo nekatere metode, smo do željenega delovanja prišli z uporabo drugih metod. Tako smo naprimer metodo `DrawArc` razreda `GraphicsEx` naredili na sledeč način:

```
GraphicsPathEx gp = new GraphicsPathEx();
gp.AddArc(rect, startAngle, sweepAngle);
DrawPath(pen, gp);
gp.Dispose();
```

Podobne prijeme kot pri `DrawArc` smo uporabili še pri kopici drugih metod (`DrawEllipse, DrawRectangle,` itd.).

4.3.3 Izdelava grafičnega uporabniškega vmesnika

Naša mobilna aplikacija je verzija `Sledilca` za dlančnik, zaradi česar mora tudi uporabniški vmesnik ostati skladen tistim v `Sledilcu` namizne aplikacije. Kljub temu, da že imamo izdelan uporabniški vmesnik, ki ga moramo

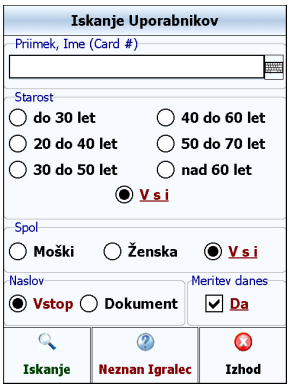


Slika 14: Osnovno okno na dlančniku

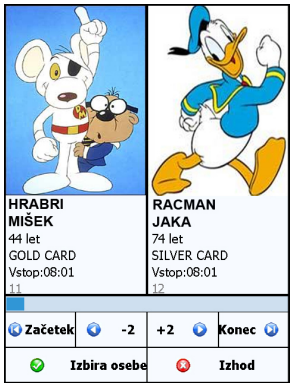
Slika 15: Menu z akcijskimi gumbi, ter sliko in podatki o igralcu

prenesti na dlančnik, pa sama izvedba ni trivialna. Prvi problem je velikost zaslona, saj na majhen zaslon dlančnika težje spravimo toliko informacij kot jih imamo na navadnem monitorju. Ta problem bi lahko rešili z čim manjšimi vizualnimi gradniki, ki bi nam zagotavljali nemoteno delo. Vendar tu naletimo na drug problem. Dlančnik ima zaslon občutljiv na dotik, preko katerega

rokujemo z aplikacijo. Za nemoteno rokovanje je tako potrebno zagotoviti dovolj velike vizualne gradnike, da je mogoče upravljanje aplikacije tudi s prsti in ne samo z pisalom. Ker nam to preprečuje manšanje vizualnih gradnikov moramo sprejeti kompromis, da manjkrat uporabljene informacije postavimo v neko podrobnejše okno ali meni, ki se nam pokaže le na zahtevo. Tako smo gumbe z akcijami našega osnovnega okna v `Sledilcu` (sledilec: slika 7, dlančnik: slika 14) postavili v poseben menu (slika 15), ki se prikaže ob kliku na gumb `Akcije`. Pri drugih oknih aplikacije smo z lahkoto postavili vse potrebne vizualne gradnike.



Slika 16: Iskanje gostov na dlančniku



Slika 17: Prikaz gostov na dlančniku

Slika 18: Zaključevanje meritve na dlančniku

4.4 Komunikacija s strežnikom

Sledilec komunicira z strežnikom preko .NET Remoting-a. Ker ta v .NET kompaktnem ogrodju ni podprt, je bilo potrebno poiskati alternativno rešitev. Kot najbolj primerna tehnologija so se pokazale spletne storitve. Pri vpeljavi spletnih storitev smo imeli več različnih možnosti implementacije. Prva možnost je uporaba klasičnih `asmx` spletnih storitev, kot druga možnost pa se nam ponuja uporaba `wcf` spletnih storitev. Pri slednjih pa imamo še na izbiro objavitve na IIS (ang. Internet Information Services) strežniku ali pa jih vključimo v Windows storitve(ang. windows service). Ker imamo naš strežnik že narejen kot Windows storitev, je izbira `wcf` spletnih storitev, ki so integrirane v windows storitev, edina smotrna odločitev.

4.4.1 Izdelava spletnih storitev

Pri izdelavi spletnih storitev je najprej potrebno narediti vmesnik z opisi vseh potrebnih metod. Ta vmesnik bo storitvena pogodba (ang. ServiceContract), ki določa podprte operacije, lokacijo operacij, posebne protokole in serializacijske formate, ki so potrebni za uspešno komunikacijo z storitvjo. Vsako metodo vmesnika, za katero želimo, da bo dosegljiva operacija storitve, moramo deklarirati kot operacijsko pogodbo (ang. OperationContract). Za ta vmesnik je nato treba narediti razred, ki implementira vse metode vmesnika. V teh metodah smo v našem primeru kliciali metode, ki smo jih uporabljali pri .NET Remoting komunikaciji na namizni varianti aplikacije. Kot smo že napisali, je potrebno zagotoviti tudi gostovanje spletne storitve, kar bomo v našem primeru naredili znotraj Windows storitve. Za tako gostovanje je po-

trebno ustvariti `ServiceHost` z instanco naše spletne storitve ter željenega url naslova. `ServiceHost-u` se nastavi končno točko storitve ter izmenjavo metapodatkov če želimo avtomatsko generiranje kode na klientu.

4.4.2 Uporaba spletnih storitev

Uporaba spletnih storitev je precej lahko opravilo. Najprej je potrebno dodati sklic na spletno storitev. To naredimo tako, da v menuju projekta v Visual Studiu kliknemo `dodaj spletni sklic`, nato pa v oknu, ki se nam pojavi, vpišemo url naše spletne storitve. Ob potrditvi vnosa url-ja se nam zgenerirajo potrebni razredi za delo z izbrano storitvjo. Edina slaba lastnost spletnih storitev je hitrost, saj je znatno počasnejša od .NET Remoting-a. To se je najbolj poznalo pri pregledu zadetkov iskanih gostov (slika 17), kjer je prenos slik pri pomikanju med gosti trajal precej časa. Rešitev tega problema je asinhroni klic, s katerim smo v naprej naložili slike v pomnilnik, kar je precej izboljšalo odzivnost pomikanja med slikami.

4.5 Avtomatska nadgradnja aplikacije

Ker je razvoj potekal postopno in se aplikacija uporablja predvsem v večernih ter nočnih urah, nam avtomatsko nadgrajevanje predstavlja veliko optimizacijo distribucije nove verzije aplikacije. Avtomatska nadgradnja aplikacije je narejena tako, da ob vsakem zagonu aplikacija najprej preveri, če je trenutna verzija enaka zadnji objavljeni verziji. Če temu ni tako, se zažene aplikacija `Webdeploy`, ki je zadolžena za izvedbo nadgradnje. `Webdeploy` ob zagonu potrebuje kot vhodni argument url namestitvene datoteke. Na podlagi tega prenese namestitveno datoteko iz spletnega stražnika, jo namesti v tihem načinu (brez prikaza namestitvenih obvestil). Za tiho namestitev je potrebno imeti na napravi certifikat, s katerim je podpisana namestitvena datoteka. Po končani namestitvi `Webdeploy` zažene nameščeno aplikacijo.

Poglavje 5

Sklepne ugotovitve

Cilj diplomske naloge - izdelava mobilne aplikacije na podlagi obstoječe namizne aplikacije - je popolnoma uspel, saj se aplikacija uspešno uporablja pri sledenju igre igralcev v igralnici. Mobilna aplikacija je dobro sprejeta tudi med uporabniki, kar odraža tudi dejstvo, da je večina sledenj vnešeno prav z mobilne aplikacije.

Pri razvoju se je kot največja prednost izkazala uporaba tronivojske arhitekture pri namizni aplikaciji, saj je bila večina poslovne logike že narejene na strežniku, kar je pomenilo bistveno zmanjšan obseg programiranja pri izdelavi mobilne aplikacije. Tronivojska arhitektura nam prav tako olajša dodajanje novih funkcionalnosti, ker poslovno logiko definiramo samo na strežniku in jo lahko uporabimo tako v namizni kot v mobilni aplikaciji.

Z razvojem vizualnih gradnikov ter knjižnice za risanje smo definirali osnovo za izdelavo novih mobilnih aplikacij, saj se nam ne bo več potrebno posvečati izdelavi vizualnih gradnikov.

Možne so izboljšave pri izrisovanju vizualnih gradnikov, kjer bi bilo mogoče njihov izris precej pohitriti. Mobilno aplikacijo pa bi izboljšali tako, da bi ji dodajali nove funkcionalnosti, ki bi olajšale njeno uporabo. Primeri takih funkcionalnosti so:

- prikaz vseh aktivnih sledenj, ki bi omogočal izbiro posameznega sledenja in bi nas ob izbiri pozicioniral na igralno mizo igralca ter
- pomikanje po zgodovini izbranih miz, ki bi sledilcu omogočalo, da se pomika med do sedaj že izbranimi mizami s pomočjo smernih tipk dlančnika.

Literatura

[1] (2009) .NET Compact Framework. Dostopno na:
http://en.wikipedia.org/wiki/.NET_Compact_Framework

[2] (2009) .NET Framework. Dostopno na:
http://en.wikipedia.org/wiki/.NET_Framework

[3] (2009) .NET Framework Security. Dostopno na:
http://msdn.microsoft.com/en-us/library/aa720329(VS.71).aspx

[4] (2009) .NET Remoting. Dostopno na:
http://en.wikipedia.org/wiki/.NET_Remoting

[5] (2009) .NET Remoting Overview. Dostopno na:
http://msdn.microsoft.com/en-us/library/kwdt6w2k(VS.71).aspx

[6] (2009) Assemblies Overview. Dostopno na:
http://msdn.microsoft.com/en-us/library/k3677y81(VS.71).aspx

[7] (2009) Common Language Infrastructure. Dostopno na:
http://en.wikipedia.org/wiki/Common_Language_Infrastructure

[8] (2009) Graphics Device Interface. Dostopno na:
http://en.wikipedia.org/wiki/Graphics_Device_Interface

[9] (2009) Introduction to the .NET Framework Class Library. Dostopno na:
http://msdn.microsoft.com/en-us/library/hfa3fa08(VS.71).aspx

[10] (2009) Metadata Overview. Dostopno na:
http://msdn.microsoft.com/en-us/library/xcd8txaw(VS.71).aspx

[11] (2009) Personal digital assistant. Dostopno na:
http://en.wikipedia.org/wiki/Personal_digital_assistant

[12] (2008) The History of Windows Mobile. Dostopno na: http://www.brighthub.com/computing/windows-platform/articles/1295.aspx

[13] (2009) Web service. Dostopno na: http://en.wikipedia.org/wiki/Web_service

[14] (2009) Web services overview. Dostopno na: http://help.eclipse.org/stable/index.jsp?topic=/org.eclipse.jst.ws.doc.user/concepts/cws.html

[15] (2009) What is Mono. Dostopno na: http://mono-project.com/What_is_Mono

[16] (2009) Windows Mobile. Dostopno na: http://en.wikipedia.org/wiki/Windows_Mobile